Ministerul Educatiei, Culturii și Cercetarii al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatica

Report

Laboratory work Nr.1

Course: Operating Systems

Implemented by:

Alexandra Konjevic, FAF-213

Verified by:

Rostislav Călin

Chișinău – 2023

Subject: Writing text using assembly

Objectives:

Create a program in assembler which will print text to the screen, in the following methods:

- M1: Write character as TTY
- M2: Write character
- M3: Write character/attribute
- M4: Display character + attribute
- M5: Display character + attribute & update cursor
- M6: Display string
- M7: Display string & update cursor
- M8(optional): Print directly to video memory.

Compiled program should be used in order to create a floppy image and it should be bootable. Use this image to boot the OS in a VirtualBox VM and the text which you intended to print should appear on the screen.

Implementation:

Method 1:

```
go:
    mov AH, 0Eh
    mov AL, 57h
    int 10h
```

Figure 1. Program that writes char as TTY

This is a program that writes a character as TTY. mov AH, 0Eh - this instruction loads the value 0Eh (which is the BIOS function code for "Teletype Output") into the AH (accumulator high) register. This function code is used to indicate that we want to print a character to the screen.

Figure 2. Output of method 1

Method 2:

```
org 0x7c00

start:
    mov ah, 0Ah
    mov al, 'e'
    mov cx, 6
    int 0x10

halt:
jmp halt

times 510-($-$$) db 0

dw 0xAA55
```

Figure 3. Program to output char

First of all, the program uses the function `0Ah` to write a character. mov al, 'e' - this instruction moves the ASCII character 'e' into the AL register. The AL register is typically used for data manipulation. Next, using the register `cx` I specified how many times the character should be displayed. int 0x10 is a software interrupt instruction that triggers a BIOS video services call. It's used to perform operations related to the display, and in this case, it's being used to print the character 'e' to the screen. `halt` - this is another label that defines a point in the code. It's labeled "halt," which suggests that this is intended to be an infinite loop to stop the program from

executing further. `jmp halt` is an unconditional jump instruction that jumps to the "halt" label, creating an infinite loop. The program effectively stops here, as it repeatedly jumps to itself.

Output:

Figure 4. Output of method 2

Method 3:

```
BITS 16
ORG 0x7c00        You, in

start:
    mov ah, 0x09
    mov al, 'e'
    mov bl, 4; red color
    int 0x10

halt:
    jmp halt

times 510-($-$$) db 0
dw 0xAA55
```

Figure 5. Program to display char, attribute

This program uses the function 0x09, to display a string. Also, in this program I use the register `bl`, to specify the color of the text that will be displayed – in this case, the red color:

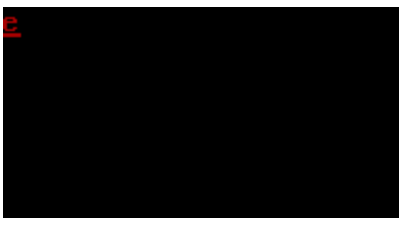

Figure 6. Output of method 3

Method 4:

```
BITS 16
ORG 0x7c00

start:
    mov ah, 0x09
    mov al, 'e'
    mov bl, 4; red color
    mov cx, 4; 4 times
    int 0x10

halt:
    jmp halt

times 510-($-$$) db 0
dw 0xAA55
```

Figure 7. Program to output char 4 times with attribute

In this program I also use the function to display a string (`0x09`). Here, the string consists of the character 'e' that is repeated 4 times (`mov cs, 4`) and the letters are of red color (`mov bl, 4`):

Figure 8. Output of method 4

Method 5:

```
BITS 16
ORG 0x7c00

start:
    mov ah, 0x09
    mov al, 'e'
    mov bl, 4; red color
    mov cx, 2; 2 times
    int 0x10

    ; cursor position
    mov ah, 02h; function to set the cursor position
    mov dx, 0x0003; row 0 col 3
    int 0x10

halt:
    jmp halt

times 510-($-$$) db 0
dw 0xAA55
```

Figure 9. Program to write char with updated cursor

This program displays the character two times, and displays it in red color. Also, here I use the function 02h to set the cursor's position. The letters occupy two columns (because there are two letters `e` displayed). That's why I set the position of the cursor in row zero and column 3:

Figure 10. Output of method 5

Method 6:

```
BITS 16
ORG 0x7c00

msg db "Hello, World!", 0
msg_len equ $-msg

start:
    mov ax, 0x1300
    mov bl, 0x02      ; Text color (green)
    mov cx, msg_len
    mov dh, 0x01      ; Row 1
    mov dl, 0x00      ; Column 0
    mov bp, msg
    int 0x10
    jmp $

times 510-($-$$) db 0
db 0x55, 0xAA
```

Figure 11. Program to write string

First of all, the program defines a message (string) with the text "Hello, World!" and a null terminator (0) at the end. It's stored in memory, and the msg label is used to reference the start of the message. `msg_len equ $-msg` calculates the length of the message by subtracting the current location ($) from the starting location (msg). It uses the equ directive to assign this value to msg_len. mov ax, 0x1300 sets the AX register with the hexadecimal value 0x1300. In this

context, it's used to set the video mode, where 0x13 corresponds to a 320x200 VGA graphics mode, and the 00 part may be a color attribute. The dh and dl line are used to indicate the row and column where the text will be displayed. `mov bp, msg` loads the BP register with the memory address of the msg string, so it points to the beginning of the message.

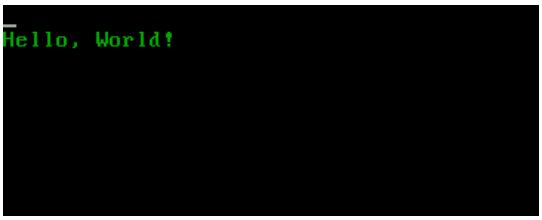

Figure 12. Output of method 6

Method 7:

```
BITS 16
ORG 0x7c00

msg db "Hello, World!", 0
msg_len equ $-msg

start:
    mov ax, 1300h
    mov al, 1; display the entire string and update cursor
    mov bl, 0x03; blue
    mov cx, msg_len
    mov dh, 0x01     ; Row 1
    mov dl, 0x00     ; Column 0
    mov bp, msg
    int 0x10
    jmp $

times 510-($-$$) db 0
db 0x55, 0xAA
```

Figure 13. Program to write string and update cursor

This program resembles the previous one, with the difference in the line `mov al, 1`, which is used to display the string and move the cursor.

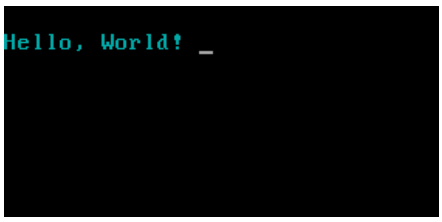

Figure 14. Output of method 7

In conclusion, this laboratory work has provided a valuable opportunity to gain practical experience in x86 assembly language programming for writing characters and strings. Throughout the course of this project, I have successfully implemented multiple programs that demonstrate our ability to interact with memory, registers, and BIOS interrupts to achieve specific tasks related to character and string manipulation.